定数ラウンドのMapReduce計算に対する計算複雑さに関する研究

著者	今澤 規真
出版者	法政大学大学院理工学研究科
雑誌名	法政大学大学院紀要．理工学・工学研究科編
巻	59
ページ	1-8
発行年	2018-03-31
URL	http://doi.org/10.15002/00021581

法政大学大学院理工学・工学研究科紀要　Vol.59(2018 年 3 月）　　法政大学

定数ラウンドの MapReduce 計算に対する計算複雑さに関する研究

A STUDY ON COMPUTATIONAL COMPLEXITY
FOR MAPREDUCE COMPUTATION IN CONSTANT ROUND

今澤規真
Norimasa IMAZAWA
指導教員　和田幸一教授

法政大学大学院理工学研究科応用情報工学専攻修士課程

MapReduce framework attracts attention very much in handling big data, and interest increases how much thing computing power of MapReduce is. In this paper, we investigate relationship between NC^k (computation time is $O(log^k n)$) and MRC^k, which is a class of problems computed by MapReduce with k rounds. As a result, we made clear that we could carry out a problem of the NC^1 by a constant round in MRC.

Key Words : MapReduce, computational complexity, MRC, NC, BP, PBP

1. はじめに

近年,情報の膨大化が進み,テラバイトやペタバイトといった人間が扱うデータ量が増えてきている. そのようなビックデータをどう処理するかという問題に対して, MapReduce という並列処理フレームワークに注目が集まっている. MapReduce は Google 社によって開発され, 使用されてきたが, そのオープンソース版である Hadoop[1]による実装例が発達してきており, Yahoo!やフェイスブックを始めとした 100 社以上で使われている[1]. MapReduce は入力として<key, value>という形の key 値対を受け取り, 出力として新たな key 値対を求める動作を複数ラウンド行う. 各ラウンドでは, key 値対に新しい key を与え(map フェーズ),与えられた key に応じて key 値対を reducer に振り分け(shuffle フェーズ), reducer で key 毎にユーザが定義した処理を実施する(reduce フェーズ).

本研究では,[2]で提唱された MapReduce の理論モデルである MRC と他の並列処理モデル(PRAM やブール回路)を比較することで MapReduce の計算能力の検証が行われている事実に着眼した.

2. 準備

本研究を進めるに当たり, 必要となる知識を説明する. まず RAM と PRAM について以下のように定義する.

定義 1

RAM とは，ランダムアクセス機械(Random Access Machine)の略称であり，一つのプロセッサがメモリを読み書きしつつ， 計算を進めるモデルの事である．

定義 2

PRAM とは並列ランダムアクセス機械(Parallel Random Access Machine)の略称であり，RAM の処理を複数にしたモデルである．

PRAM ではその性質上，複数のプロセッサが同時並行的に共有メモリ上の同じ位置にアクセスする可能性がある．そのためメモリの同じ位置への「読み取り」と「書き込み」それぞれについて，許すか禁止するかで以下の 4 つのモデルに分けることができる．

①EREW

EREW は Exclusive Read Exclusive Write の略称で，一度に 1 つのプロセッサのみがメモリへの「読み取り」と「書き込み」が行える PRAM モデルである．

②CREW

CREW は Concurrent Read Exclusive Write の略称で，「読み取り」は複数のプロセッサで行えるが，「書き込み」は一度に 1 つのプロセッサのみが行える PRAM モデルである．

③ERCW

ERCW は Exclusive Read Concurrent Write の略称で，「読み取り」は一度に 1 つのプロセッサのみが行え，「書き込み」複数のプロセッサで行える PRAM モデルである．その性質上，基本的には考慮されない．

④CRCW

CRCW は Concurrent Read Concurrent Write の略称で，複数のプロセッサが自由に読み書きを行える PRAM モデルである．

続いて，ブール回路について以下のように定義する．

ブール回路 C は n 入力と 1 出力を持ち，AND, OR, NOT によって構成される有向非巡回グラフである(n ∈ N)．入力辺が 0 のノードを入力ノード，出力辺が 0 のノードを出力ノードと呼び，それ以外のノードをゲートと呼ぶ．ゲートは AND, OR, NOT でラベル付けされる．

回路 C のノードのレベルを次のように定める．回路 C の入力ノードのレベルを 0 とする．ノード g のレベルはノード g に隣接するノードのレベルの最大値よりも 1 大きい値である．回路 C の深さ(depth)を回路 C の中で最大のノードのレベルとする．回路 C におけるレベル i の厚み(thickness)をレベル i のノード数とする．回路 C の幅(width)を回路 C におけるレベル i の厚みの中で最大のものとする．回路 C のサイズを回路 C 中の全ゲートの入力辺の総数をとする．1 つのゲートに入力される辺の数をファンインとする．

次に，複雑度クラスについて定義する．

複雑度クラスとは，ある量の計算資源を使って受理することができる言語の集合のことである．複雑度クラスの定義にあたって，最初に以下の定義 3 を示す．

定義 3

Rec_nはサイズ n の入力Xから，0 または 1 を決定する認識器であり，$\mathrm{REC} = \bigcup Rec_n$とする．$Rec_n$が$L_n = \{X | X \in L, |X| = n\}$を受理するとは，$\mathrm{X} \in L_n$であり，且つその時$R_n(X) = 1$である．

RECが言語Lを受理するとは各自然数 n に対して，Rec_nがL_nを受理するということである．定義 3 を基に認識器となる複雑度クラスをこれから順に定義していく．

定義 4　クラス SPACE

クラス SPACE(S)は，work tape のサイズが S のチューリング機械である認識器で受理される言語の集合である．チューリング機械とは，長さが無制限で記号を読み書きできるテープ，記号の読み書きを行うヘッド，そのヘッドとテープのシークを制御する機能をもつ有限オートマトンである．

定義 5 クラスNC^k（$k \geq 0$の定数）

クラスNC^kは深さ$\mathrm{O}(\log^{\mathrm{k}}\mathrm{n})$，多項式サイズ，ファンインが定数に制限された AND, OR, NOT ゲートからなるブール回路である認識器で受理される言語の集合である．クラスNC^kの部分集合としてクラスAC^k，クラスTC^kが存在する．

定義 6 クラスAC^k（$k \geq 0$の定数）

クラスAC^kは深さ$\mathrm{O}(\log^{\mathrm{k}}\mathrm{n})$，多項式サイズ，ファンインの制限がない AND, OR, NOT ゲートからなるブール回路である認識器で受理される言語の集合である．

定義 7 クラスTC^k（$k \geq 0$の定数）

クラスTC^kはクラスAC^kの認識器に多数決ゲート(入力数の過半数が 1 なら 1, そうでなければ 0 を出力するゲート)を付け加えた認識器によって受理される言語の集合である.

定義より, $\mathrm{AC}^k \subseteq \mathrm{TC}^k$であり, $\mathrm{TC}^k \subseteq \mathrm{NC}^{k+1}$が成り立つ[3].

定義 8

クラス$\mathrm{CRCW}^{\mathrm{k}}$, $\mathrm{EREW}^{\mathrm{k}}$, $\mathrm{CREW}^{\mathrm{k}}$は多項式個のプロセッサと$\mathrm{O}(log^k n)$時間の CRCW, EREW, CREW である認識器によって受理される言語の集合である.

3. MapReduce モデル(MRC)

MapReduce 計算モデルについて定義する[2].

定義 9

mapper μは, key 値対<k, v>を入力として,key 値対, $<k_1, v_1>, <k_2, v_2>, \cdots$ を出力する.

reducer ρは, 各 key k とその値のリスト$<\mathrm{v}_1, \ldots, \mathrm{v}_n>$を入力として受け取り, 各 key k と新しい値のリスト$<\mathrm{v'}_1, \ldots, \mathrm{v'}_n>$を出力する.

MRC マシンの実行動作は以下の通りである.

1. $\mathrm{U}_{\mathrm{r}-1}$は最後のラウンドからの key 値対のリストを示す. mapper μ_rへは$\mathrm{U}_{\mathrm{r}-1}$の key 値対を入力とする.(r=1 の場合は, 入力の key 値対を指す.)
2. shuffle and sort は key によって値をグループ化する. $\mathrm{V}_{\mathrm{k,r}} = \{\mathrm{k}, (\mathrm{v}_{\mathrm{k},1}, \mathrm{v}_{\mathrm{k},2}, \ldots)\}$とする.
3. reducer ρ_rへ各$\mathrm{V}_{\mathrm{k,r}}$を割り振り, $\mathrm{U}_{\mathrm{r}} = \mathrm{U}_k \rho_r(\mathrm{V}_{\mathrm{k,r}})$となる.

上記の各 r については, map ステップ, shuffle ステップ, reduce ステップの 3 つから構成される.

MRC とは MapReduce Class の略称であり,以下の条件を満たす.

$\mathrm{MRC}^{\mathrm{i}}$は認識器であり, mapper μ_rと reducer ρ_rで構成される列$\langle \mu_1, \rho_1, \mu_2, \rho_2, \ldots, \mu_R, \rho_R \rangle$で受理される言語の集合である. そして$\mu_1$の出力が$\rho_1$の入力となり, ρ_1の出力がμ_2の入力となり, ρ_Rの出力によって$\mathrm{MRC}^{\mathrm{i}}$が言語を受理するかが決定する. mapper と reducer は容量が$\mathrm{O}(\mathrm{N}^{\mathrm{c}})$であり, それぞれ$\mathrm{O}(\mathrm{N}^{\mathrm{c}})$個ずつ存在する. N は入力サイズで, N は$\mathrm{U}_0$(1 ラウンド目)の入力となる key 値対に必要となる総ビット数である. c は$\frac{1}{2} \leq \mathrm{c} < 1$を満たす.(本論文では$\mathrm{c} = \frac{1}{2}$に固定する) また, $\mathrm{R} = \mathrm{O}(\log^{\mathrm{i}} \mathrm{N})$である.

4. 先行研究

これまでに様々な複雑度クラスや MRC の定義を示してきたが, ここでは従来の結果として主だった先行研究を紹介する. まず今までに挙げた複雑度クラスについて次の事が分かっている[4].

定理 1

$$\mathrm{NC}^k \subseteq EREW^k \subseteq CREW^k \subseteq CRCW^k = \mathrm{AC}^k \subseteq \mathrm{NC}^{k+1}$$

次に紹介する問題集合はMRC^0, 即ち MapReduce 計算を定数ラウンド行うことで計算が可能である.

(1) o(logn)領域のチューリング機械によって計算できる問題集合[4].
(2) AND, OR, NOT からなるファンイン制限のない論理回路によって定数時間で計算できる問題集合[5].

本論文では(2)の結果を拡張し, (2)を真に含むファンイン制限のある AND, OR, NOT からなる論理回路によって o(logn)時間で計算できる問題集合(NC^1)が定数ラウンドの MapReduce 計算(MRC^0)で計算できること, つまり$\mathrm{NC}^1 \subseteq \mathrm{MRC}^0$となることを示す.

5. 分岐プログラム(BP)

クラスNC^1がMRC^0の部分集合であることを証明するのにあたって, 分岐プログラムを本研究では取り上げた.先ずは[6]を基にした BP の定義を行う.

定義 10

BP とは分岐プログラム(Branching Program)の略称で, 以下の性質を持つ決定木である.

①ただ 1 つの source を持つ

②各 node は高々2 つの出力辺を持つ

③出力辺が 2 の node は, 一方の辺がブーリアン変数x_iによってラベル付けされ, もう一方はブーリアン変数$\overline{x_i}$よってラベル付けされる.

④sink(出力辺が 0 の node)は 0 か 1 でラベル付けされる.

BP は認識器でもある. つまり, ある分岐プログラム P がサイズ n の入力 X を受理するということは, 言語$L_n = \{X | X \in L, |X| = n\}$でラベル付けされた辺を経由して, sink が 1 になるということである.

分岐プログラムを使った先行研究について紹介する．

(1) 定数幅の BP はNC^1の回路でシミュレートできる[7]．

(2)ファンインが 2，深さが$\mathrm{O}(\log n)$のNC^1回路で識別できる任意の言語は，幅が 5 で，多項式サイズの BP でも識別ができる[7]．

(1), (2)より，BP は NC^1を模倣することができる．そこで本研究では,定数幅の置換を行う BP，w-Permutation Branching Program(以下 w-PBP と表記)をMRC^0で解くことにより，$\mathrm{NC}^1 \subseteq \mathrm{MRC}^0$となることを示した．

その前に w-PBP の定義を行う[7]．

定義 11　w-PBP (w は定数)

w-PBP は幅が w の置換分岐プログラム(Permutation Branching Program)の略称で，$< x_b, f, g >$という命令の列から成っている．x_bは入力変数 $(b \geq 0)$で，関数 f, g は [w] から [w] への関数を表す．([w] は置換で，[w] = {1,⋯,w}) そして$x_b = 1$なら f，$x_b = 0$なら g を実行する．w-PBP によって言語が受理されるとは言語L_nに対して，置換の集合Qが決まり，入力 X に対して，w-PBP が置換$q \in Q$を計算する時，かつその時に限り$X \in L_n$となることである．

6. w-PBP を MRC でシミュレート

まず,w-PBP を MRC でシミュレートできるように入力を<key, value>の形に変換する．変換後の記述を以下に定義する．

定義 12

$$\langle l, (x_b, f, g) \rangle$$

BP の行番号を示すlを key に，入力変数x_b $(0 \leq b \leq n-1)$と関数 f, g を value とする．

以下は入力サイズ N の定義である.入力として入る w-PBP プログラムの各行を表すのに必要なビット数を B，そのプログラムが t 行で構成され，プログラムへの入力を D ビットとすると，tB+D の容量が必要になる(n との関係は,プログラムへの入力は入力変数とその値から表現でき，入力変数は n 個，値は logn で表せるため D=O(nlogn)，1 行 B ビットで表せる w-PBP が n の多項式行分続くため定数 k を用いて $\mathrm{tB} = \mathrm{O}(n^k)$である)．そこで N = tB+D と設定し，定数αを用いて mapper 全体の容量をαN とする．また個数は mapper, reducer どちらも$\sqrt{N}$個とし，各 reducer の容量は定数βを用いて$\beta\sqrt{N}$ と設定する．(α, βは定数のため MRC モデルで示した容量$\mathrm{O}(\sqrt{N})$，個数$\mathrm{O}(\sqrt{N})$を満たす)

シミュレートの流れを説明する．w-PBP を 1 つの reducer で処理するのは不可能なので，w-PBP を$\sqrt{N}$個の部分プログラムに分割し，分割した部分プログラムを各 reducer 内で計算し，その結果を 1 つの reducer に集めて w-PBP の出力を求める．図 1 に大まかな概略を示す．

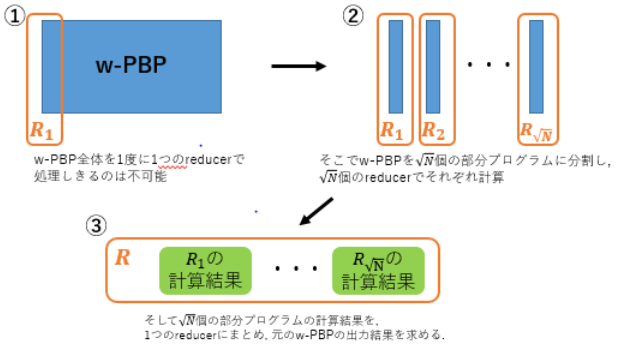

図 1　w-PBP を MRC でシミュレートする流れ

$\sqrt{N}$個の部分プログラムを計算するために，入力変数の値の割り振りを上手く行う必要がある．そこで各部分プログラムで出現した入力変数の数をカウントし，プレフィックスサムを求めることでカウントした入力変数の数が reducer の容量$\sqrt{N}$を越さないような区切りの値を求め，その区切りに従って各 reducer 内で値の割り振りを行い，部分プログラムを計算し，最後に結果をまとめることで出力を得る．

補題 1. w-PBP を分割した部分プログラムに出現する入力変数の値は MRC モデルにおいて 7 ラウンドで求められ，値を付与できる．

1 ラウンド目

入力: w-PBP プログラム

出力: 各部分プログラム中に出現した変数の種類

mapper の動作

w-PBP を$\sqrt{N}$個ずつに分割する．

reducer の動作

各 reducer $\mathrm{R_j}(0 \leq j \leq \sqrt{N}-1)$が担当する部分プログラム中に出現する$x_b$を新しい key に，新しい value を検出した reducer と定数 1 をペアとした$(\mathrm{R_j}, 1)$として出力する．

mapper で必要な容量は w-PBP プログラムのみが入り

きればよいので，α＝１で収まりきる．reducer に必要な容量は各 reducer で求めた入力変数の種類と reducer のインデクスなので$\sqrt{N}\log\sqrt{N}$，β＝２あれば十分である．

2〜4 ラウンドの計３ラウンドを使って，1 ラウンド目で求めた各入力変数の出現数のプレフィックスサムを求める．以下はサブルーチンとして，key 値対<key_b，1>(0 ≤ b ≤ N − 1 の整数)を受け取った時にkey_bのプレフィックスサムを求める流れを紹介する．

1 ラウンド目

入力: key 値対 $\langle \mathrm{key}_b, 1\rangle$

出力:key 値対 $\langle b, Psum_b\rangle$

mapper の動作

bdiv$\sqrt{N}$を行い，この key 値対を$\sqrt{N}$個に分割する．

reducer の動作

各$\mathrm{R_j}$は，key_b毎に value の値を集計し，その値を$count_b$とする．さらに各 reducer 内で求めた$count_b$からプレフィックスサムを求め，$Psum_b$とする．新しい key をb，新しい value を$Psum_b$として出力する．

mapper に必要な容量は$\langle \mathrm{key}_b, 1\rangle$全てなので b，reducer に必要な容量は$\langle b, Psum_b\rangle$を出力するため，blogn である．ここで図 2 に$Psum_b$の値を示す．

$$Psum_0 = count_0 \qquad R_0\text{で求まるプレフィックスサム}$$
$$Psum_1 = count_0 + count_1$$
$$Psum_2 = count_0 + count_1 + count_2$$
$$\vdots$$
$$Psum_{\sqrt{N}-1} = count_0 + \cdots + count_{\sqrt{N}-1}$$
$$Psum_{\sqrt{N}} = count_{\sqrt{N}} \qquad R_1\text{で求まるプレフィックスサム}$$
$$Psum_{\sqrt{N}+1} = count_{\sqrt{N}} + count_{\sqrt{N}+1}$$
$$\vdots$$
$$Psum_{2\sqrt{N}-1} = count_{\sqrt{N}} + \ldots + count_{2\sqrt{N}-1}$$
$$Psum_{2\sqrt{N}} = count_{2\sqrt{N}}$$
$$\vdots$$
$$Psum_{N-1} = count_{(\sqrt{N}-1)\sqrt{N}} + \cdots + count_{N-1}$$

図２　各$Psum_b$の値

図 2 に示したように$\mathrm{R_0}$で求めた$count_0$から$\mathrm{count}_{\sqrt{N}-1}$までのプレフィックスサムは正しく求まっているが，それ以降の reducer では，それ以前のプレフィックスサムの情報がないため，不完全なものとなっている．そこで３ラウンド目で穴抜けになったプレフィックスサムを求め，4 ラウンド目で正しいプレフィックスサムを求める．

2 ラウンド目

入力: key 値対 $\langle z\sqrt{N}-1, Psum_{z\sqrt{N}-1}\rangle\ (1 \le z \le \sqrt{N})$

出力: key 値対 $\langle z, Compsum_z\rangle$

mapper の動作

全ての key 値対に*という共通の key を与える．

reducer の動作

$Psum_{z\sqrt{N}-1}$の値からプレフィックスサムを求め，$Compsum_z$とする．新しい key をz，新しい value を$Compsum_z$として出力する．

mapper に必要な容量は$\langle z\sqrt{N}-1, Psum_{z\sqrt{N}-1}\rangle$全てなので$(\sqrt{N}-1)logn$, reducer に必要な容量も$\langle z, Compsum_z\rangle$を出力するため，$(\sqrt{N}-1)logn$である．

ここで図 3 に$Compsum_z$の値を示す．

$$Compsum_1 = count_0 + \cdots + count_{\sqrt{N}-1}$$
$$Compsum_2 = count_0 + \cdots + count_{2\sqrt{N}-1}$$
$$\vdots$$
$$Compsum_{\sqrt{N}-1} = count_0 + \cdots + count_{(\sqrt{N}-1)\sqrt{N}-1}$$
$$Compsum_{\sqrt{N}} = count_0 + \cdots + count_{N-1}$$

図３　各$Compsum_z$の値

図 3 より，$Compsum_1 = Psum_{\sqrt{N}-1}$であり，これは前ラウンドで求めた$x_0$から$\mathrm{x}_{\sqrt{N}-1}$までのプレフィックスサムなので正しいのは明らかである．そしてz ≥ 2の場合は，

$$Psum_{z\sqrt{N}-1} = \sum_{q=1}^{z\sqrt{N}-1} count_q - \sum_{q=1}^{(z-1)\sqrt{N}-1} count_q \quad \cdots ①$$
$$Compsum_z = \sum_{q=1}^{z} Psum_{q\sqrt{N}-1} \quad \cdots ②$$

①，②より$Compsum_z$が正しいプレフィックスサムであることを証明できる．

3 ラウンド目

入力: $\langle \mathrm{b}, \mathrm{Psum_b}\rangle$ $\langle \mathrm{z}, \mathrm{Compsum_z}\rangle$

出力: $\langle \mathrm{b}, \mathrm{Psum'_b}\rangle$

mapper の動作

$\langle \mathrm{b}, \mathrm{Psum_b}\rangle$は$bdiv\sqrt{N}$を行い，$\sqrt{N}$個ずつ割り振る．

$\langle \mathrm{z}, \mathrm{Compsum_z}\rangle$は処理を行わない．

reducer の動作

$Psum_b$の値からプレフィックスサムを求める．$\mathrm{R_1}$から$\mathrm{R}_{\sqrt{N}-1}$では求めたプレフィックスサムに$Compsum_z$の値を足して，正しいプレフィックスサムを求める．ここで求めたプレフィックスサムを$\mathrm{Psum'_b}$とし，key を b，value を$\mathrm{Psum'_b}$として出力する．

mapper に必要な容量は$\langle b, Psum_b\rangle$が b 個，$\langle z, Compsum_z\rangle$が$\sqrt{\mathrm{N}}-1$個あるので blogn +$(\sqrt{N}-1)$logn,

reducer に必要な容量は，$\langle b, Psum_b\rangle$ が $\sqrt{N}$ 個，$\langle z, Compsum_z\rangle$が 1 個なので，$\sqrt{N}$logn+logn である．ここで図 4 に正しく求めたプレフィックスサム$Psum'_b$の値を示す．

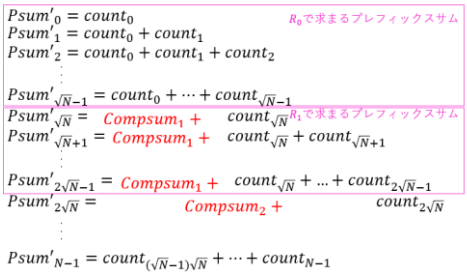

図 4 各$Psum'_b$の値

上記のプレフィックスサムを求めるサブルーチンを利用して，1 ラウンド目で求めた各入力変数の出現数のプレフィックスサムを求めることができる．なおサブルーチンの 3 ラウンド目の reducer において，R_1から$R_{\sqrt{N}-1}$では$\sqrt{N}$個の key 値対$\langle bdiv\sqrt{N}, (b, Psum_b)\rangle$と 1 つの key 値対$\langle z, Compsum_z)\rangle$を受け取るため$\sqrt{N}$を越してしまうが，定数βを β=2 とすれば十分である．

次に入力変数x_bの出現回数のプレフィックスサムの値から$\sqrt{N}$毎の区切りを見つける．またサブルーチンで 3 ラウンド使用しているため，次ラウンドを 5 として記載している．

5 ラウンド目

入力: key 値対 $\langle b, Psum'_b\rangle$

出力: key 値対<(d’, x_b)， >

mapper の動作

$\langle b, Psum'_b\rangle$を$\sqrt{N}$個に分割する．分け方は b の値が小さい順に$\sqrt{N}$個ずつ各 reducer に分けていくが，$R_0$~$R_{\sqrt{N}-2}$は次の reducer が担当する key 値対の最初の組を含むように分割する．その様子を以下の図 5 に示す．

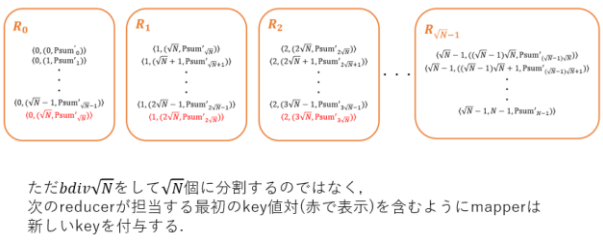

図 5　5 ラウンド目の mapper の key 値対の割り振り方

reducer の動作

$Psum'_b div\sqrt{N}$を求め，その結果を d’とする．d’の値が同じになった key 値対の中で最も b の値が大きいものに対して，key が d’とx_bの組，value が空白の key 値対<(d’, x_b), >を生成する．そして，R_0は生成した全ての key 値対を，R_1~$R_{\sqrt{N}-1}$は生成した key 値対の中で一番 d’の値が小さい key 値対以外の key 値対全てを出力する．

各 reducer に次の reducer が担当する key 値対の最初の組を含ませて，R_1~$R_{\sqrt{N}-1}$で生成した生成した key 値対の中で一番 d’の値が小さい key 値対を出力させないことで，reducer 間で d’の値が変化するしない関わらず，$\sqrt{N}-1$個の区切りを見つけることができる．

mapper に必要な容量は$\langle b, Psum_b\rangle$が b 個,あるので blogn, reducer に必要な容量は，$R_{\sqrt{N}-1}$以外の reducer で$\langle b, Psum_b\rangle$が$\sqrt{N}$+1 個入力されるので，$(\sqrt{N}+1)$logn となり，reducer の容量$\sqrt{N}$を上回る．しかし，定数βを β=2 とすれば十分である．

6 ラウンド目

入力: 1 ラウンド目の出力<x_b, $(R_J, 1)$>と<(d’, x_b)， >

出力: $\langle d', (x_b, R_j)\rangle$

mapper の動作

<x_b, $(R_J, 1)$>はjを新しい key にして，<(d’, x_b)， >は全ての reducer に割り振る．

reducer の動作

入力変数x_bのインデクス b の値と key 値対<(d’, x_b), >の b の値(区別のため b‘と表記)を順に比較し，インデクス b が b’の値を下回った時の d’の値を新しい key に，新しい value を(x_b, R_j)として，区切りと入力変数と reducer のインデクスを持った key 値対を出力させる．

mapper に必要な容量は<x_b, $(R_J, 1)$>が高々N 個，<(d’, b)， >が$\sqrt{N}$-1 個あるので,N(log$\sqrt{N}$)+$\sqrt{N}$-1(log)，reducer に必要な容量は,<x_b, $(R_J, 1)$>が$\sqrt{N}$個，<(d’, x_b)， >が$\sqrt{N}$-1 個あるので$\sqrt{N}$(log$\sqrt{N}$)+($\sqrt{N}$-1)logn となり，mapper と reducer の容量$\sqrt{N}$を上回る．しかし，定数α,βを共に 4 とすれば十分である．

7 ラウンド目

入力: $\langle d', (x_b, R_j)\rangle$ $\langle Psum_b, (x_b, num_b)\rangle$

出力: $\langle R_j, (x_b, num_b)\rangle$

num_bはx_bの値であり,その値は 0 または 1 である．

mapper の動作

key 値対$\langle d', (x_b,\ R_j)\rangle$は処理を行わない． key 値対$\langle Psum_b, (x_b, num_b)\rangle$は$Psum_b div \sqrt{N}$を行い $\sqrt{N}$個に振り分ける．

reducer の動作

新しい key をR_j，新しい value を(x_b, num_b)にして，R_jが必要としている入力変数x_bの値num_bを結びつける．

mapper に必要な容量は，$\langle d', (x_b,\ R_j)\rangle$が高々N 個，$\langle Psum_b, (x_b, num_b)\rangle$が n 個から，N(2log$\sqrt{\text{N}}$+logn)+n(logn + 1)，reducer に必要な容量は，$\langle d', (x_b,\ R_j)\rangle$が高々$\sqrt{\text{N}}$個，$\langle Psum_b, (x_b, num_b)\rangle$も高々$\sqrt{\text{N}}$個なので ,$\sqrt{\text{N}}$(2log$\sqrt{\text{N}}$+logn)+$\sqrt{\text{N}}$（logn+1）となり，mapper と reducer の容量$\sqrt{\text{N}}$を上回る．しかし，定数α,βを共に 4 とすれば十分である．

補題 2. 入力変数の値が割り当てられた BP を入力して，w-PBP の置換を 2 ラウンドの MRC 求めることができる．

以下に 2 ラウンドで行う処理を示す．

1 ラウンド目

入力: w-PBP プログラムと$\langle R_j, (x_b, num_b)\rangle$

出力: $\langle R_j,\ \sigma_{R_j}\rangle$

mapper の動作

w-PBP は$\sqrt{N}$個ずつに分割し，key 値対$\langle R_j, (x_b, num_b)\rangle$は新しい key を j とする．

reducer の動作

各 reducer で出現する入力変数x_bの値num_bが分かるので，num_bの値から部分プログラムにおける置換結果σ_{R_j}を求め，新しい key をR_j，新しい value をσ_{R_j}とする．

mapper に必要な容量は， w-PBP プログラムで tB，$\langle R_j, (x_b, num_b)\rangle$が高々N 個から，tB+N(log$\sqrt{\text{N}}$+logn+1)，reducer に必要な容量は$\sqrt{\text{N}}$分割した w-PBP プログラムと，$\langle R_j, (x_b, num_b)\rangle$が高々$\sqrt{\text{N}}$個なので，t B/$\sqrt{\text{N}}$ + $\sqrt{\text{N}}$（log$\sqrt{\text{N}}$+logn+1）となり，mapper と reducer の容量$\sqrt{\text{N}}$を上回る．しかし，定数α,βを共に 4 とすれば十分である．

2 ラウンド目

入力: $\langle R_j,\ \sigma_{R_j}\rangle$ $\langle *,\ [w]\rangle$

出力: $\langle *, result\rangle$

mapper の動作

key 値対を一つの reducer に集めるため共通の key ∗を与える．

reducer の動作

reducer で$[w]$に対して，R_jの j の値が小さい順に置換関数σ_{R_j}を行う．その結果を result とし，key を∗, value を result として出力する．

result は[w]に置換関数σ_{R_j}を行った結果である．また置換関数σ_{R_j}は$\sqrt{N}$個の置換関数 f または g を行った結果であり，いずれもlの値が小さい順に処理を行ってきた．そのため$result$は行番号の順に動作した元の PBP と同じ出力結果を示している．

本論文では MRC の計算能力を調べる上で，NC との関係性を研究した．そして，NC^1の回路を定数幅の BP でシミュレートできる点に注目し，定数幅の BP を MRC でシミュレートし，定数ラウンドで行えることを明らかにした．従って，補題 1，補題 2 より定理 5 を証明できる．

定理 5. $\text{NC}^1 \subseteq \text{MRC}^0$

7. 今後の課題

NC^1についてはMRC^0でシミュレートできることを示したが，依然として MRC とNC^kとの関係性は触れられていない．また$\text{SPACE}(O(\log n))$とMRC^0の関係性も明らかではないことが今後，研究を行う上での課題であり，目標となる．

謝辞

本研究を進めるに当たり，ご指導頂いた和田幸一教授に深く感謝致します．また日々，有益な情報と数々の刺激を与えて下さった研究室の皆様に感謝致します．

参考文献

[1] Apache Software Foundation, Hadoop Wiki: PoweredBy. http://wiki.apache.org/hadoop/PoweredBy, 2012.

[2] Karloff, H., Suri, S., Vassilvitskii, S. “A model of computation for mapreduce” SODA 2010, pp. 938-948. Society for Industrial and Applied Mathematics, Philadelphia (2010)

[3] David S. JOHNSON, “A Catalog of complexity

Classes", In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity, 67-161, 1990.

[4] Richard M.KARP, Vijaya RAMACHANDRAN, "Parallel Algorithms for Shared-Memory Machines" In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity, 869-941, 1990.

[5] Benjamin Fish, Jeremy Kun, Adam D. Lelks, Lev Reyzin and Gyorgy Turan, "On the Computational Complexity of MapReduce" , Proceedings of 29th International Symposium on Distributed Computing (DISC 2015) , Lecture Notes in Computer Science, 9363, 1-15 2015.

[6] 間々田 剛史, "MapReduce 計算の並列複雑さに関する研究", 法政大学大学院理工学・工学研究科修士論文 Vol.57, (2016 - 03)

[7] Allan Borodin, Danny Dolev, Faith E. Fich, Wolfgang Paul, "Bounds for Width Two Branching Programs", SIAM J. Comput, 15(2), 549-560.

[8] David A. Barrington, "Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1", Journal of Computer and System Sciences 38, 150-164 (1989)